In [ ]:

```
### Futures Trend-Following & Risk Management

## Niraj Neupane


Project 3: Futures Trend-Following & Risk Management (CTA-style) – ETF Proxies

What this script does:
- Downloads daily adjusted close prices for liquid ETF proxies (SPY, QQQ, TLT, GLD,
- Builds a time-series momentum (trend-following) strategy
- Uses volatility targeting + leverage cap for position sizing
- Applies a simple portfolio-level drawdown stop (go flat after breach)
- Produces plots + saves outputs (equity curve + metrics) to ./outputs

Requirements:
- Python 3.9+
- pip install numpy pandas matplotlib yfinance

Notes:
- Uses ETF proxies rather than futures contracts to keep it simple and reproducible
- This is a research backtest (no transaction costs/slippage by default). You can a
```

In [2]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Optional dependency check for yfinance (clear error message)
try:
    import yfinance as yf
except ModuleNotFoundError as e:
    raise ModuleNotFoundError(
        "Missing dependency 'yfinance'. Install it with: pip install yfinance"
    ) from e


# ------------------------------
# Config
# ------------------------------
TICKERS = ["SPY", "QQQ", "TLT", "GLD", "USO"]  # ETF proxies for major futures
START_DATE = "2010-01-01"
END_DATE = None  # None = up to today

LOOKBACK_DAYS = 252          # ~12 months momentum lookback
VOL_WINDOW_DAYS = 63         # ~3 months vol estimation
TARGET_VOL_ANNUAL = 0.10     # target 10% annualized vol
LEVERAGE_CAP = 3.0           # cap gross leverage per asset
DD_STOP = -0.20              # portfolio drawdown stop (e.g., -20% => go flat)
RF = 0.0                     # risk-free rate assumed 0 for Sharpe

OUTPUT_DIR = "outputs"
PLOT_STYLE = "seaborn-v0_8"
```

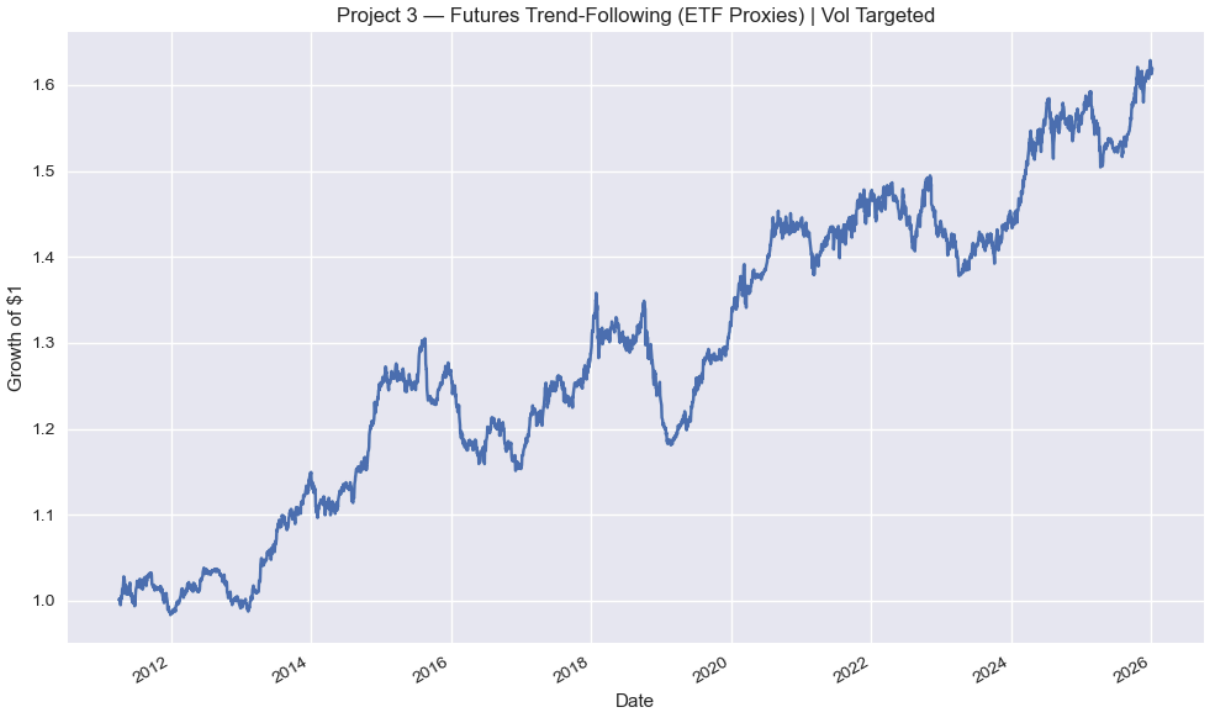
```python
# ----------------------------
# Helpers
# ----------------------------
def download_prices(tickers, start, end=None):
    """
    Download adjusted close prices using yfinance.
    Returns a DataFrame indexed by date with columns=tickers.
    """
    data = yf.download(tickers, start=start, end=end, auto_adjust=True, progress=Fa
    if isinstance(data, pd.DataFrame) and "Close" in data.columns:
        prices = data["Close"].copy()
    else:
        # Sometimes yfinance returns a single-level frame for single ticker
        prices = data.copy()

    # Ensure DataFrame format
    if isinstance(prices, pd.Series):
        prices = prices.to_frame(name=tickers[0])

    prices = prices.dropna(how="all")
    prices = prices.ffill().dropna()
    return prices


def compute_ts_momentum_signal(prices, lookback_days):
    """
    Time-series momentum signal:
    signal_t = sign( price_t / price_{t-lookback} - 1 )
    Shifted by 1 day to avoid lookahead (use yesterday's signal for today's trade).
    """
    mom = prices.pct_change(lookback_days)
    signal = np.sign(mom)
    signal = signal.shift(1)
    return signal


def compute_vol_target_weights(returns, vol_window_days, target_vol_annual, leverag
    """
    Vol targeting per asset:
    weight_{t,i} = target_vol / vol_{t,i}
    where vol_{t,i} is rolling annualized volatility.
    We cap absolute weights by leverage_cap.
    """
    rolling_vol = returns.rolling(vol_window_days).std() * np.sqrt(252)
    weights = target_vol_annual / rolling_vol
    weights = weights.replace([np.inf, -np.inf], np.nan)
    weights = weights.clip(lower=-leverage_cap, upper=leverage_cap)
    return weights


def max_drawdown(equity_curve):
    """
    Compute max drawdown from an equity curve (Series).
    """
    peak = equity_curve.cummax()
    dd = equity_curve / peak - 1.0
```

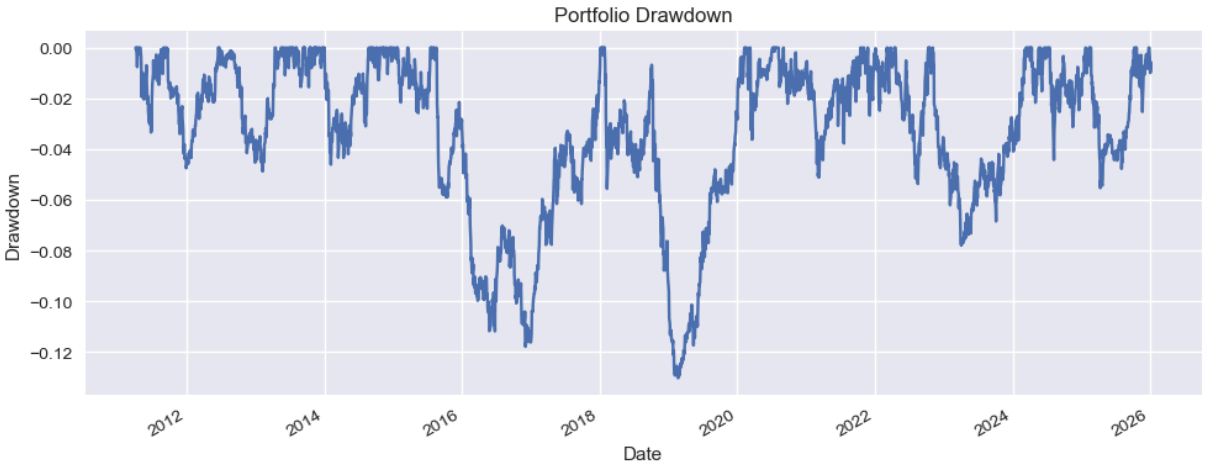
```
    return dd.min(), dd


def perf_stats(strategy_returns, rf=0.0):
    """
    Basic performance stats for daily returns series.
    """
    ann_ret = strategy_returns.mean() * 252
    ann_vol = strategy_returns.std() * np.sqrt(252)
    sharpe = np.nan
    if ann_vol and ann_vol > 0:
        sharpe = (ann_ret - rf) / ann_vol

    eq = (1 + strategy_returns.fillna(0)).cumprod()
    mdd, _ = max_drawdown(eq)

    # Win rate & avg win/loss
    wins = strategy_returns[strategy_returns > 0]
    losses = strategy_returns[strategy_returns < 0]
    win_rate = (strategy_returns > 0).mean()
    avg_win = wins.mean() if len(wins) else np.nan
    avg_loss = losses.mean() if len(losses) else np.nan

    return {
        "Annual Return": ann_ret,
        "Annual Volatility": ann_vol,
        "Sharpe (rf=0)": sharpe,
        "Max Drawdown": mdd,
        "Win Rate": win_rate,
        "Avg Daily Win": avg_win,
        "Avg Daily Loss": avg_loss,
        "Observations": int(strategy_returns.dropna().shape[0]),
    }


# ------------------------------
# Main
# ------------------------------
def main():
    # Style + output folder
    plt.style.use(PLOT_STYLE)
    os.makedirs(OUTPUT_DIR, exist_ok=True)

    # 1) Load prices
    print("Downloading prices...")
    prices = download_prices(TICKERS, START_DATE, END_DATE)
    print("Prices shape:", prices.shape)
    print(prices.head())

    # 2) Compute daily returns
    returns = prices.pct_change().dropna()

    # 3) Build signals (trend direction)
    signal = compute_ts_momentum_signal(prices, LOOKBACK_DAYS)

    # Align indices (signal & returns)
```

```
signal = signal.reindex(returns.index).dropna(how="all")
returns = returns.reindex(signal.index).dropna(how="all")

# 4) Vol targeting weights
vol_weights = compute_vol_target_weights(
    returns=returns,
    vol_window_days=VOL_WINDOW_DAYS,
    target_vol_annual=TARGET_VOL_ANNUAL,
    leverage_cap=LEVERAGE_CAP,
)
vol_weights = vol_weights.reindex(returns.index)

# 5) Final position weights: direction * vol-target weight
#    If signal is +1 => long; -1 => short; 0 => flat
weights = signal * vol_weights

# Remove early NaNs (from rolling windows)
valid_mask = weights.notna().all(axis=1)
weights = weights.loc[valid_mask]
returns = returns.loc[valid_mask]

# 6) Strategy daily returns (equal-weight across assets after weighting)
#    Each asset return multiplied by its weight; then average across assets.
#    (You can also use sum instead of mean if you want "capital deployed" inter
strat_asset_returns = weights * returns
strat_returns = strat_asset_returns.mean(axis=1)

# 7) Equity curve + drawdown stop
equity = (1 + strat_returns.fillna(0)).cumprod()

mdd_before, dd_series = max_drawdown(equity)
print(f"Max drawdown before stop: {mdd_before:.2%}")

# Apply a simple drawdown stop:
# When drawdown <= DD_STOP, set returns to 0 going forward (flat).
stop_trigger = dd_series <= DD_STOP
if stop_trigger.any():
    first_stop_date = stop_trigger.idxmax()  # first True index
    print(f"Drawdown stop triggered on: {first_stop_date.date()} (DD <= {DD_STO
    strat_returns_stopped = strat_returns.copy()
    strat_returns_stopped.loc[first_stop_date:] = 0.0
    equity_stopped = (1 + strat_returns_stopped.fillna(0)).cumprod()
    strat_returns_used = strat_returns_stopped
    equity_used = equity_stopped
else:
    print("Drawdown stop not triggered.")
    strat_returns_used = strat_returns
    equity_used = equity

# 8) Metrics
stats = perf_stats(strat_returns_used, rf=RF)
metrics_df = pd.DataFrame([stats])

print("\nStrategy performance (after drawdown stop logic):")
for k, v in stats.items():
    if isinstance(v, float):
```

```
            # nice formatting
            if "Rate" in k or "Return" in k or "Volatility" in k or "Drawdown" in k
                print(f"{k}: {v:.2%}")
            else:
                print(f"{k}: {v:.4f}")
        else:
            print(f"{k}: {v}")

    # 9) Plots
    fig = plt.figure(figsize=(10, 6))
    equity_used.plot()
    plt.title("Project 3 — Futures Trend-Following (ETF Proxies) | Vol Targeted")
    plt.ylabel("Growth of $1")
    plt.xlabel("Date")
    plt.tight_layout()
    plt.show()

    # Drawdown plot
    peak = equity_used.cummax()
    dd = equity_used / peak - 1.0

    fig = plt.figure(figsize=(10, 4))
    dd.plot()
    plt.title("Portfolio Drawdown")
    plt.ylabel("Drawdown")
    plt.xlabel("Date")
    plt.tight_layout()
    plt.show()

    # 10) Save outputs
    equity_out = equity_used.to_frame(name="equity_curve")
    equity_path = os.path.join(OUTPUT_DIR, "trend_equity.csv")
    metrics_path = os.path.join(OUTPUT_DIR, "trend_metrics.csv")

    equity_out.to_csv(equity_path)
    metrics_df.to_csv(metrics_path, index=False)

    print("\nSaved outputs:")
    print(" -", equity_path)
    print(" -", metrics_path)


if __name__ == "__main__":
    main()
```

```
Downloading prices...
Prices shape: (4026, 5)
Ticker             GLD        QQQ        SPY        TLT         USO
Date
2010-01-04  109.800003  40.341591  85.027962  56.763184  322.160004
2010-01-05  109.699997  40.341591  85.253052  57.129749  323.279999
2010-01-06  111.510002  40.098251  85.313034  56.364964  327.760010
2010-01-07  110.820000  40.124325  85.673180  56.459827  325.760010
2010-01-08  111.370003  40.454575  85.958290  56.434544  327.440002
Max drawdown before stop: -13.03%
Drawdown stop not triggered.

Strategy performance (after drawdown stop logic):
Annual Return: 3.41%
Annual Volatility: 5.21%
Sharpe (rf=0): 0.6537
Max Drawdown: -13.03%
Win Rate: 54.59%
Avg Daily Win: 0.0024
Avg Daily Loss: -0.0026
Observations: 3711
```






```
Saved outputs:
 - outputs\trend_equity.csv
 - outputs\trend_metrics.csv
```